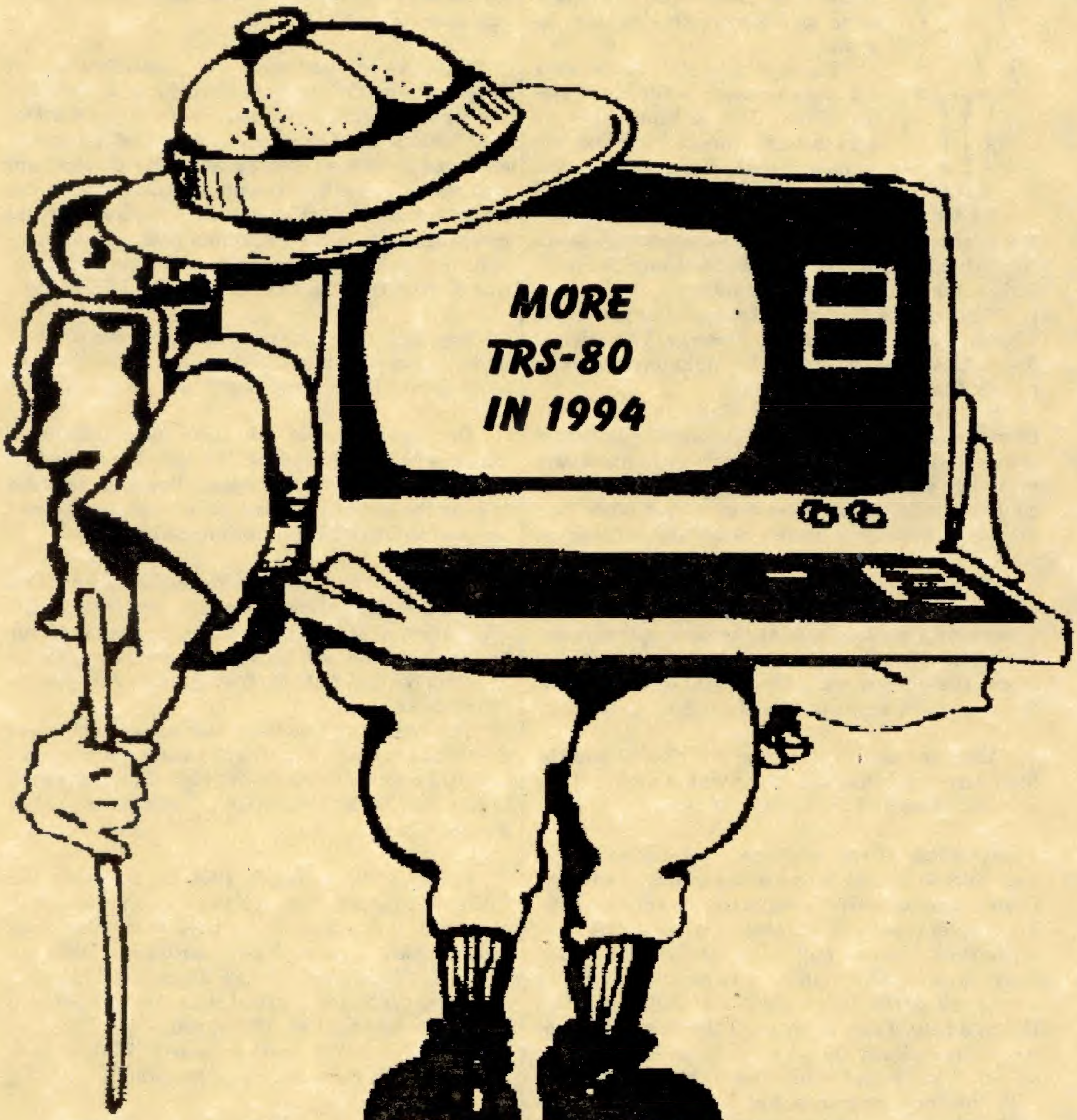
TRSTimes

Volume 7. No. 1 - Jan/Feb 1994 - $4.00



LITTLE ORPHAN EIGHTY

Immediately after mailing out the Nov/Dec issue of TRSTimes, the world around us went up in flames and, thanks to CNN, Topanga Canyon Boulevard became known the world over. The fires were devastating and many people lost their homes and possessions, including some of Hollywood's rich and famous.

Topanga Canyon Boulevard's northern boundary is the Simi Freeway (Hwy. 118) in Chatsworth. It goes through Canoga Park and into Woodland Hills. Here, it crosses the Ventura Freeway (Hwy. 101), and immediately the flatlands ends, and the boulevard then twists and turns its way south through the Santa Monica mountains until it finally ends at the Pacific Ocean in Malibu.

TRSTimes is located on Topanga Canyon Blvd., 1 block north of the Ventura Freeway. The original fire was started by person(s) unknown approximately 4 miles to the south of us.

We were lucky, indeed. While several of my friends and acquaintances had to leave their homes behind and flee to safety, we merely spent a few days nervously watching the direction of the fire. Thank goodness, it kept going away from us and, other than having to breathe a little smoke, we suffered no harm.

We were worried about our friend, Jim King, who lives in Fernwood up in Topanga Canyon. It was good to see him safe at the computer meeting the Friday after the fire was no longer a threat. Jim graces these pages with his account of what it was like to be right in the middle of danger.

Allen Jacobs explains how the TRS-80 and the hard drive communicate, along with a listing of the port addresses and the function of each.

For those of you who enjoy playing games on your TRS-80, Danny Myers walks through two of the famous Infocom adventures, Starcross and Deadline. I was never that enthused about Starcross, but being a Sherlock Holmes buff, I naturally spent many hours on Deadline. Suffice it to say, Baker Street was much better off without me! But now, with Danny's help, I can go through the program to see where my deductive abilities failed me - and you can as well. I know that Danny has logged many hours with the Infocom games, but I still marvel at the skill with which he solves a puzzle. Oh, well!

TRSTimes has covered Basic and Assembly Language, but we have given very little space to the C language. Why? Mainly because 'Ye faithful Editor' is not well versed in that dialect of computer speak. Not to worry though, Frank Slinkman is an accomplished C programmer (actually he is an accomplished programmer - period), and he brings us an article and program that will expand Pro-MC's (from Misosys) abilities.

This issue presents two installments of 'Programming Tidbits' from Chris Fara of Microdex. The first is a reprint of the article from the Nov/Dec 1993 issue where I, for various inexcusable reasons, managed to make it unreadable. My sincere and humble apologies. The second installment is for the Model 4 and is full of good stuff. Programmers and non-programmers alike will enjoy being able to automate screen layout. Also included is a good explanation of of LS-DOS 6.3's use of the USR11 function.

Roy Beck takes us through the various printers he has used with his TRS-80's and, in the process, talks about a famous local hardware whiz.

Our machines do not have 'sound-blasters', midi's, and 'what-have-you', but they are never-the-less capable of producing sound. However, in order to hear the sound you must have a speaker hooked up - and Kelly Bates tells us how he has done it.

I am fascinated with the Model I emulator. This issue I am presenting a patch to both the MAKFIL/CMD and MAKFILE4/CMD programs from our last issue so they will be able to correctly create an emulator 'virtual disk' file from double density Newdos/80 disks.

The articles in this issue took up so much space that I did not have room to write about the programs that will convert an emulator 'virtual disk' file back in to a real Model I disk. Well, I will get to that in March.

I would like to begin 1994 by thanking the TRSTimes contributors for their fine articles and programs. It is wonderful to have that many dedicated people still caring for our machines. Unlike the Apple II, Commodore 64 and Atari, the TRS-80 is still being supported in grand style. And it is because of you, the contributors. Thank you.

Also Thank You to the readers. Without you there would be no need for new material.

And now....... *Welcome to TRSTimes 7.1*

TRSTimes magazine

Volume 7. No. 1 - Jan/Feb 1994

PUBLISHER-EDITOR
Lance Wolstrup
CONTRIBUTING EDITORS
Roy T. Beck
Dr. Allen Jacobs
TECHNICAL ASSISTANCE
San Gabriel Tandy Users Group
Valley TRS-80 Users Group
Valley Hackers' TRS-80 Users Group

TRSTimes is published bi-monthly by TRSTimes Publications. 5721 Topanga Canyon Blvd., Suite 4. Woodland Hills, CA 91367. U.S.A. (818) 716-7154.

Publication months are January, March, May, July, September and November.

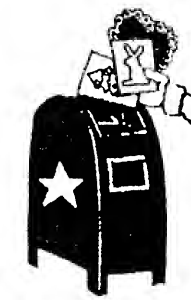
1994 subscription rates (6 issues):
UNITED STATES & CANADA:
$20.00 (U.S. currency)

EUROPE, CENTRAL & SOUTH AMERICA:
$24.00 for surface mail or $31.00 for air mail. (U.S. currency only)

ASIA, AUSTRALIA & NEW ZEALAND:
$26.00 for surface mail or $34.00 for air mail. (U.S. currency only)

Article submissions from our readers are welcomed and encouraged. Anything pertaining to the TRS-80 will be evaluated for possible publication. Please send hardcopy and, if at all possible a disk with the material saved in ASCII format. Any disk format is acceptable, but please note on label which format is used.

THE MAIL ROOM



ALLWRITE, DOTWRITER FONTS AND THE EMULATOR

I am usually not excited about running software, but the TRS-80 emulator, running on a clone, is exciting. It means that I do not have to use a new word processor — and now most of my TRS-80 software will run on my PC.

I am using a wordprocessor written by Chuck and Glenn Tessler in the 80's for use on a TRS-80 (AllWrite), and I can now use it on both my antique TRS-80 Model III and my clone, so I do not have to adjust my thinking when I change machines. An accompanying program called DotWriter, written by William Mason, will let you format all your text in fancy fonts. The distributors of DotWriter (Prosoft) went out of business with about 290 fonts, but I have personally generated another 140 or so that are designated Shareware and are free for the price of distribution from Mickey Mepham in Virginia (*see ad on page 21. Ed.*) Font generation is an on-going project of mine, so there will probably be more releases in the future.

Since AllWrite runs so neat in the clone, the font generation software ought to also. I have not checked it out yet, but if so, I may generate the new fonts on my clone for TRS-80 emulator use.

Using AllWrite in the emulator I have to turn on my printer after entering the emulator environment since I operate it in the Star mode and not the IBM mode. Other than that I maintain a DATA floppy on one of the virtual drives to save my editing. The floppy is 80 tracks, but if the directory entry shows less, then I have to be sure the directory shows enough free space or AllWrite will not write to it. Maybe the author (Jeff Vavasour) can change something, although I know he wants to maintain Model I standards. Being pretty good I can work around it, but other users may not do as I do.

So, how did I move all these files from the TRS to the clone? My method — in III mode on the TRS-80 (Model 4), copy the files to a single density disk. Boot in the Model 4 native mode and copy the files from the single density disk to the Model 4 disk. RESET the files. Run Hypercross and copy the files to a 360K PC formatted disk in drive one.. Then you are ready to boot the clone, follow the emulator instructions for making a virtual disk and install the TRS programs on your clone. And Yes, it was not all clear, and I did request some help, but I persisted.

I have never enjoyed using software as I now do in the emulator and, wonder of wonders, it works as advertised! There is one quirk that I am working around. The keyboard can be in Model I mode or PC mode. I want to use the arrow keys and am unable to do so, as my DOS book does not tell me what the scan codes are (perhaps I need help understanding DOS books). I further believe we should use the keys on the PC even when they a located in a different place. Not complaining, just critiquing. The arrows keys are used a lot when making fonts.

Kelly Bates
Oklahoma City, OK

CRYPTO/BAS

I am learning more and more about my Model 4, and I have type in almost all the programs that are in TRSTimes publications Jan-Dec. Most programs, which I type and hard-copy, run without a hitch. However, I am having trouble with the program "CRYPTO/BAS". I have typed it and hard copied it as printed in 6.4, page 11. I have compared the hard copy with the listing in the magazine. They are identical. The program will ENCODE, but will not DECODE. When I attempt to run the program, I get an error message "FOR WITHOUT NEXT IN LINE 330". Please help.

Also, the printout of Vigenere's code (grid) has an error in it. Column Z as taken from the magazine, reads Z z a b c s e f, etc. It should read: Z z a b c d e f, etc.

In closing I want to tell you how much I enjoy your magazine and I feel that I am learning an awful lot from it.

Hooked on the TRS-80
Anthony E. Luczak
Armada, MI

I plead guilty to the grid error. It should read, as you point out, Z z a b c d e f, etc. The program listing, however, is correct as printed. Just to be sure, I typed in the code from the magazine, and it works correctly. The error message indicates that you have a loop error. Check lines 330 and 340. Do you have FOR Z= and NEXT Z? Check lines 350 and 420. Do you have FOR Z= and NEXT Z? Check lines 360 and 380. Do you have FOR X= and NEXT X? Check lines 390 and 410. Do you have FOR Y= and NEXT Y? I will bet that you have omitted one or more of these commands.

Ed.

GETTING CONTROL OF HARD DISK I/O

as explained to Allen Jacobs

Vernon Hester once challenged TRS-80 owners to write a hard disk driver for Multidos. At that time, I realized that while I have a "foggy" idea about assembly language, I had no idea about how to write a driver since I really didn't know what a hard disk driver does. As it turned out, nobody accepted his challenge. So, he wrote a hard disk driver himself.

A while ago, when Roy Beck was investigating the inner workings of the Radio Shack hard drives and their controller boards, I was trying to acquire a basic concept of what he was doing. Namely, I couldn't quite discern exactly how hard drives operate. Anyone who has read the many articles Roy has written on the subject knows that he is very generous with his knowledge. Fortunately, I am privileged to be a member in some of the same computer clubs he attends.

While he would discuss the various aspects of hard drives as he does in his articles, I always thought that I was missing some essential information that would unify all the various elements of hard drive operation into a single concept. It's what I call getting a clear "vision" of the system.

For me, the missing element was how the computer actually communicates with the hard drive. I reasoned that if I could know what messages were being passed between the computer and the drive, I could clarify the picture of the system I had in my mind. Those messages are carried through ports between the computer and the hard disk itself via the hard disk controller and host adapter.

One evening I called Roy and asked him if he knew what was going through each of the ports. He said that he did. However, when I asked him if there was a single published listing available of all the related ports and what they do, his answer was: "Not that I know of. Not in one place."

Roy then took over an hour while I was on the phone and sifted through his notes. The result of that search is the listing you will see later in this article. Over the next couple of months, I sorted and formatted the information, constantly checking with Roy for its accuracy and completeness in order to make the listing as you finally see it.

I believe that I learned a great deal in compiling it. I think that if you look at it closely, you will develop a clearer idea of just how a hard drive functions. Then, you can almost picture in your own mind how a hard disk driver might work.

THE ELEMENTS

To review, the system hardware consists of the computer, the host adapter, the hard disk controller, and the hard disk. The software consists of the DOS and the hard disk driver.

The computer communicates with peripheral storage devices through its ports (the Model 1 floppy drive being an exception). The Z-80 has 256 ports, numbered 0-255. Ports are not actual physical electronic passageways. Rather, think of the ports as another block of 256 one byte "memory addresses" separate from the addressable memory in the computer's "core". The Z-80 has these addresses located elsewhere on the computer's bus. Theoretically, all of the ports can be tied to one single "plug" attached to the bus.

How are the ports distinguishable from each other? That is done by their unique addresses. Part of each IN instruction the Z-80 executes contains a second byte, the port address. That byte tells the Z-80 that the next data byte it detects on the bus is coming from the external device with that specifically assigned port address.

How are the ports assigned? Each external device connected to the bus has an electronic circuit that is activated by only one specific bit pattern corresponding to its assigned address. When it detects a match, it turns on and outputs its current byte. Otherwise, it continues to store its current byte, if it has one.

What if more than one device responds? You would then have a port conflict and resulting chaos. This is the only way the TRS-80 has to distinguish from which port a byte of information is coming. Otherwise, all data bytes on the bus would appear to be coming from any and all ports at the same time. Actually, there is another way the Z-80 has of dealing with ports that the TRS-80 doesn't use called an interrupt, however, we won't discuss that here.

The same process holds for any information to be passed OUT of the machine on the bus. Basically, the Z-80 announces that the next data byte to appear on the bus is intended only for the device on the bus that is activated by the port address contained in the current instruction. All the other devices with other port addresses should thus ignore it.

In hex, the ports are numbered from 00-FF, which correspond to decimal 0-255. Radio Shack was kind enough to assign ports 0-127 to users and after market manufacturers, promising that they would never assign a port number below number 128 to any Radio Shack device, to prevent conflicts as much as possible. This explains the port assignments we will encounter later on.

Why can't this information go directly to the hard drive? Because not all hard drives are constructed in the same way. They have different capacities, numbers of platters, numbers of heads, numbers of cylinders, numbers of sectors, numbers of bytes per sector, and some other characteristics, that the TRS-80 running on a given DOS simply can't automatically know about. Don't forget, these drives were built to run on any given computer made at that time.

Therefore, something has to be placed in between these two devices that can translate between hard disk I/O and the TRS-80 bus. That something is called the hard disk controller. Yes, there are also a number of those out there. Thus, we have to settle on one type if we are to make any headway in understanding the overall process. Since it is the most common and most elegant for its time. Radio Shack settled on the Western Digital controller. So, did Roy. If you own a Radio Shack Hard Drive as I do, so did you.

That's not the end of it, however. The Western Digital 1000 Series Disk Controllers were likewise designed to connect to any computer on the market. Therefore, an accommodation specific to the TRS-80 bus has to be included to insure that the bus connections between the computer, the controller, and the hard disk all go where they are supposed to and are interpreted correctly at every point. That accommodation is called the host adapter. Roy Beck has investigated and written about this device, to no small extent.

Now, with all that equipment constructed, installed, and operating, how does the DOS tell the hard disk what to store and how to store it? Also, how do you tell the DOS whether it was stored correctly through all this equipment and that a given requested byte, previously stored on the disk is ready for retrieval? That is the job of a piece of software called the Hard Disk Driver.

While I don't propose to write such a driver program, it is interesting to understand the information it is dealing with. If you want a hard disk driver, Roy Soltoff has written "RS Hard". RS Hard, and thus, Roy Soltoff know all about the information in this article.

Since the Radio Shack Host Adapter is actually built into the Western Digital Controller Boards used in Radio Shack Hard Drives, a number of port assignments and other lines have been changed to accommodate the Radio Shack convention of not assigning ports below number 128 (80 Hex). Others have been added. Both Roys know which they are. I don't.

THE SYSTEM

The listing is organized into a simple hierarchy. The PORT NAMES section lists a descriptive name for each of the ports used by the Radio Shack Hard Drive. Not all of the ports are named. Not all of the names are official. Hopefully, all of them are descriptive. The port addresses are listed in hex. The hex port addresses C0-CF correspond to ports number 192-207. Personally, I think they make more sense in hex.

The PORT DETAILS describe every last known byte that passes through the port and what it means. The Read-Write convention is what information the TRS-80 Reads from the hard disk system, and what it Writes to the hard disk system.

Some details not apparent in the listing are the meaning of Write Precompensation, Write Current Reduction, and Ramped Seek. The distance between the magnetic bits on the inner tracks of any disk are spaced closer together than they are on the outer tracks. This means that a strong write current could begin to effect the value of adjacent bits on the inner tracks. Reduction of the current in the head during a write reduces this problem.

Another form of write precompensation was also made by advancing or delaying the writing of adjacent bits that may interfere with each other. The time alteration is by fractions of a single timed pulse within the controller. The time alterations are created by taking advanced or retarded taps of the data to be written from a delay line located in the hard drive circuitry. The fractional separation of pulses in this manner makes the data easier to read.

On earlier drives, reduction of the write current on the inner tracks and write precompensation were not done automatically. The controller had to tell the drive to reduce the write current and initiate write precompensation, above a given track number.

Ramped Seek was available on later drives to allow faster access when the head travels between cylinders. In Ramped Seek mode, the number of cylinders to move in or out could be given at a higher rate of speed than the stepping rate of the drive and it would just store the number. Earlier drives would lose the track count. Thus, Ramp Seek can only be used if your drive can handle it. Otherwise the number of cylinders the head should move should be written to the drive as they are executed.

Beyond these terms, most of the items in the listing should be self descriptive (famous last words). If you look at what information is passing through the ports you can see how the whole process works.

The Radio Shack Hard Disk is accessed through ports C0H - CFH. These assigned ports are found at the host adapter on the outside of the drive. They do not correspond to the ports of the Western Digital 1001/1010 type hard disk controller, because the host adapter is interposed between the external connector and the controller itself.

THE RADIO SHACK HARD DISK CONTROLLER PORT ASSIGNMENTS

(from Roy Beck's distillation of his own notes and information)

PORT NAMES

R = READ W = WRITE

R W	PORT#	NAME/FUNCTION
R	C0	Write protect register
R W	C1	Control register
R	C2	Device present register
	C3	Ignore - used on Western Digital board - but not RS hosts
	C4	Ignore - same as above
	C5	Ignore - same as above
	C6	Ignore - same as above
	C7	Ignore - same as above
R W	C8	Data register
R	C9	Error register
W	C9	Write precompensation
R W	CA	Sector count
R W	CB	Sector number
R W	CC	Cylinder number (LSB)
R W	CD	Cylinder number (MSB)
R W	CE	Sector size/drive#/head#
R	CF	Status register
W	CF	Command register

PORT DETAILS

R	C0	WRITE PROTECT REGISTER
	bit 7	write protect switch of master drive 0
	bit 6	Write protect switch of slave drive 1
	bit 5	Write protect switch of slave drive 2
	bit 4	Write protect switch of slave drive 3
	bit 3	No connection (floats to zero?)
	bit 2	No connection (floats to zero?)
	bit 1	Hard disk write protect logic
	bit 0	Interrupt request

in the above register:
0 = write protect is not on., 1 = write protect is on

R W	C1	CONTROL REGISTER
	bit 7	
	bit 6	

		bit 5	
		bit 4	Soft reset
		bit 3	Device enable
		bit 2	Wait enable (wait state enable)
		bit 1	Set during a write
		bit 0	Set during a read?
R	C2		DEVICE PRESENT REGISTER always bit pattern 0000 0001
		bit 0	is tied to +5 volts (logical 1) always
	C3		IGNORE (used on Western Digital board, but not with RS host)
	C4		IGNORE (same as above)
	C5		IGNORE (same as above)
	C6		IGNORE (same as above)
	C7		IGNORE (same as above)
R W	C8		DATA REGISTER passes the actual data during reads & writes
R	C9		ERROR REGISTER
		bit 7	bad block detect
		bit 6	CRC error data field
		bit 5	CRC error ID field
		bit 4	ID not found
		bit 3	no meaning
		bit 2	aborted command
		bit 1	track 000 error
		bit 0	DAM not found *lower bit set = more serious error*
W	C9		WRITE PRECOMPENSATION REGISTER contains the cylinder number with which to begin precompensation, divided by 4 (cyl#/4)
R W	CA		SECTOR COUNT the number of sectors/cylinders to place during format. This value must be re-written as it is decremented during formatting.
R W	CB		SECTOR NUMBER the sector number to read or write to (or the one that has just been read or written to).
R W	CC		CYLINDER NUMBER the low byte (lsb)
R W	CD		CYLINDER NUMBER the high byte (msb) (1024 max)
R W	CE		SECTOR SIZE/DRIVE#/HEAD#
		bit 7	0 (always)
		bit 6	sector size (0-4) (128, 256, 512 or 1024 bytes)
		bit 5	sector size
		bit 4	drive select (0-3) (4 drives may be controlled)
		bit 3	drive select
		bit 2	head number (0-7) (a drive may have up to 8 heads)
		bit 1	head number
		bit 0	head number
R	CF		STATUS REGISTER
		bit 7	busy (if set, no other bits are valid)
		bit 6	ready
		bit 5	write fault
		bit 4	seek complete
		bit 3	data request
		bit 2	NC (ignore)
		bit 1	NC (ignore)
		bit 0	error (if set, error register C9 is valid)
W	CF		COMMAND REGISTER

COMMANDS	7	6	5	4	3	2	1	0	HEX
1 track 0 restore	0	0	0	1	r3	r2	r1	r0	10
2 seek track	0	1	1	1	r3	r2	r1	r0	70
3 read sector	0	0	1	0	0	0	0	0	20
4 write sector	0	0	1	1	0	0	0	0	30
5 format track	0	1	0	1	0	0	0	0	50

(BITS: 7–0)

restore = to return to track 0 (the reference track)

R0 - R3 -> STEPPING RATE

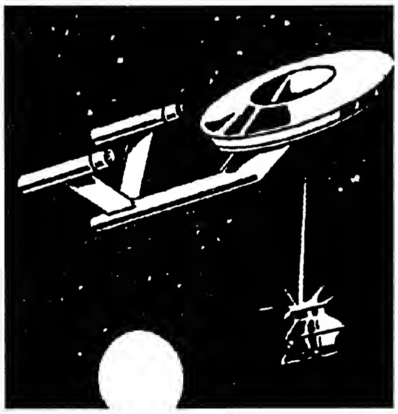
	r3	r2	r1	r0		
0	0	0	0	0	=	10 uSEC (RAMPED SEEK)
1	0	0	0	1	=	0.5 mSEC
2	0	0	1	0	=	1.0 mSEC
3	0	0	1	1	=	1.5 mSEC
4	0	1	0	0	=	2.0 mSEC
5	0	1	0	1	=	2.5 mSEC
6	0	1	1	0	=	3.0 mSEC
7	0	1	1	1	=	3.5 mSEC
8	1	0	0	0	=	4.0 mSEC
9	1	0	0	1	=	4.5 mSEC
10	1	0	1	0	=	5.0 mSEC
11	1	0	1	1	=	5.5 mSEC
12	1	1	0	0	=	6.0 mSEC
13	1	1	0	1	=	6.5 mSEC
14	1	1	1	0	=	7.0 mSEC
15	1	1	1	1	=	7.5 mSEC

D = DMA OR PROGRAMMED I/O MODE
D=0 -> programmed I/O mode (only mode we use)
if D=1 -> DMA mode

BEAT THE GAME

by Daniel Myers

STARCROSS



Ah, outer space! No dank and dusty dungeons here....but you can be sure that the puzzles are no easier than they were in the Great Underground Empire! So, settle in and get ready for lift-off.

There you are, floating around space in your ship, alarm bells ringing in your ears. Obviously, something's about to happen. Get the tape library, then get up and go Starboard into the Bridge. Push the red button, which will shut off the alarm bell, and read the screen. This will tell you which object on your map is the one to head for.

Now, there's no way I can tell you the exact coordinates, as the destination changes from game to game. However, it isn't hard to figure out what they are. Once you've done that, sit down in the control couch and fasten the belt. Now you have to enter the course into the computer, which is done as follows: Computer, range is x, theta is y, phi is z. The computer will ask you to confirm the new course, which you do by saying: Computer, confirm new program. After that, you're off!

And now it's time for the hallmark of all Infocom games: Waiting! You'll sit in the couch, and wait until you arrive at the alien ship and you are captured by it <however, you can enjoy the verbose descriptions while you wait>. Once your ship is down on the dock, unfasten the belt, get up, and go Starboard into the storage room. Get the suit, put it on, then get the line. Head Portwards back to the bridge.

Fun times with airlocks begin now. Open the inner door, go out, close the inner door, open the outer door, and go out. Get used to doing that, because you'll be doing it again, and again! So, now you're on the Red Dock, and there's a strange-looking sculpture here. Closer examination, and a little thought, shows that it's a representation of the solar system.

Aha! Could it be...? You press the fourth bump, and strange things happen. Press the small bump, and a black rod appears...get the rod, and the outer airlock door opens!

Okay, go inside <close outer, open inner, etc.>, and you are in a Red Hall. At this point, you might want to save the game, and just wander around, doing some mapping. Actually, I recommend that you do so, since there will be a point in the game where I won't be able to give you specific directional instructions.

Now that you've checked out the territory a bit, restore to your original entry point. You're ready for the great rod hunt....because the object of the game is to activtate and control the artifact, which is done via different colored rods. Right now, you want the red one, so, go North, then West into the Room on Ring Two. From there, North into the Zoo, and East into the rat-ant cage.

The red rod is part of the nest, and you just can't reach over and get it. This is one of the very few times in an Infocom where violence is necessary: Throw the tape library at the nest, which will be smashed. While the rat- ants look at you in terror, grab the red rod and the tape library.

Now it's time for the yellow rod, so head along West, South, West, which will bring you to the Blue Hall. Go South once and you're at the Blue Airlock. Open the door, go down, close inner door, etc. When you reach the Blue Dock, go Aft until you come to the Spider-like alien. It's quite intelligent and even friendly. Give it the tape library, and in return you will get the yellow rod. Take that, and make your way back to the Blue Hall.

You have two rods, and you will be using them now. From the Blue Hall, go up, and you're in the Grasslands. Go South to the Thin Forest, open the hatch there, and go down into the Repair Room. Put the yellow rod in the yellow slot. That turns on the lights in the Yellow Hallway. Put the red rod in the SECOND red slot. Make sure it's the second; this will provide a breathable atmosphere for you.

Now, get the metal square, and go up, then North, then down again to the Blue Hall. From there, West to the Yellow Hall and Yellow Airlock. You know the drill by now, but there's an extra feature this time: You will have to try to open the outer door

twice <it's balky>. Also, while you're in the Yellow Airlock, pick up the basket; it will come in very handy!

Once on the Yellow Dock, tie the line to the suit and then to the hook. Head Portwards, get the pink rod, put it in the basket, then go back Starboard to the dock, untie yourself, and return to the Yellow Hall.

Now, it's time for the blue rod. Go South twice, then East once. You are in a laboratory with a mysterious silver globe floating in mid-air. Inside the globe, although you can't see it now, is a blue rod. It's easy to get, however. Take the two disks off the wall. Put one on the floor, and one under the globe. It doesn't matter which way you do it, the result will be the same. Put the basket on the globe, then turn the dial to 4. Ta-da! The basket suddenly appears on the disk on the floor, with the blue rod! Turn the dial to 1, then get both disks, the basket, and the blue rod. Put the rod in the basket. In fact, put all rods in the basket when you get them.

Okay, there's still plenty of rods to collect, so let's keep moving! Head West, then North four times to the end of the hallway, then West to the Room on Ring One, and South from there into the Computer Room. Open the panel on the computer, then insert the metal square into the slot. Turn on the computer, and you will get a gold rod. Don't worry about all the displays; they aren't important to you.

Now comes more waiting. What you're waiting for this time is the mechanical "mouse" that collects trash. So, move around until it makes an appearance. As soon as it does, drop one of the disks <either one>. The mouse will pick it up. After that, you must follow the mouse around until it disappears into a secret door in one of the rooms. There are several different rooms where the mouse can do this, so you *must* follow it.

Wait there until it reappears and leaves, wait a little longer to make sure it won't come back, then drop the other disk on the floor and step on it. Zap!! You're in the Garage! <Hurray for transporter disks!>. Pick up the disk, then empty the trash bin <yuck!> until you find the green rod. Go North and you will be in the Room on Ring Four.

You are now in the Room on Ring Four. Now, this is why you had to do some mapping on your own: You must get the other disk you dropped, and there's no telling exactly in which room that was. So, you must explore on your own until you find the disk. Once you've done that, make your way to the Blue Hall where the airlock is. From there, go North twice, then West into the Observatory. Drop off one of the disks, then hike along East, South, East, East, South, East into the Weasel Village, and then East once more to the Village Center. Wait around a short while, and the Weasel chief will appear.

He will indicate that he wants your space suit. That's no problem, since the air will remain breathable, and you don't need the suit anymore. So, give it to him. Then, when he wants to give you something in return, point to the brown rod he wears around his neck. He will give it to you, and start to leave. Follow him! <Think of all this following around as good practice for "Deadline.">

Continue to follow him, until you arrive at the Center of the Warren. Then climb down the ladder to the Green Airlock <yep, another one!>, and do the usual job with the doors. From the dock, go West to the Umbilical, then West again to the Cargo Hold. Pick up the visor fragment, then go Forward into the Control Room of the wrecked ship. Move the skeleton, and you will find a violet rod. Now, drop the disk on the floor, and step on it. If you attempt to leave the way you came, the Weasels will kill you for disturbing their shrine.

So, now you've materialized in the Observatory, and it's time to pick up another rod: Look at projector through visor. Aha! A clear rod. Sneaky, huh? Get the rod, drop the visor, then move along East, then South three times to the Melted Spot, then West into the Weapons Deck. Get the genuine Ray-gun, and look inside it.

Sonuvagun! A silver rod! Get that, then East and North, and up to the Grasslands, 'cause it's time to get this show on the road. Now, trek on South twice to the Dense Forest, then East to the base of the tree. Climb the tree, all the way up to the top, then jump to the Drive Bubble.

Insert the silver rod into the slot, then enter the bubble. There's a white rod here; get that and put it into the white slot. Under no circumstances should you insert the black rod into the black slot!! That will shut everything down! Okay, the drive mechanism has been activated; now you have to make the thing move. So, out, and up to the top of the Bubble, and....jump! Isn't floating in air fun? However, you still have some things to do yet, so shoot the gun at the Drive Bubble three times, which will bring you to the Control Bubble. Go Down, then put the gold rod in the slot, and enter the Bubble. Inside, you will find the slots for the remaining rods. Put each rod in the slot of its own color.

Now, at last, you're ready to bring the artifact to life! Touch the large pink square, and the scene in the small one will change to show the inner solar system. Now touch the brown spot until a picture of Earth appears.
Press the violet one until a ellipse shows <it *must* be an ellipse!>, then press the green spot, and flashing lights appear. And last, but certainly not least, the final move: Touch the blue spot, which activates everything, and brings the alien ship safely to Earth!

Of course, it isn't over yet! That final remark by the alien sounded a little ominous....I have a feeling you'll be heading out again into space sooner than you might think!

DEADLINE

Here is the solution for solving Infocom's mystery game, Deadline.

From the front path of the Robner's estate, go [N] to the front door of the house. Type "OPEN DOOR" and go [N] into the Robners' house. From inside the door, go [N,E] and type "CLIMB STAIRS" twice (or you can just go [U,U]) to get to the second level of the estate. From there, go [W,W,W,W,N] to the library where you will start the first of a series of Sherlock Holmes-type activities.

In Deadline, you need to establish the motive and method for the murder beyond all reasonable doubt before you can arrest the guilty party. If you don't have an air-tight case, the jury will acquit the defendant. It is here in the library where we go about establishing the method by which poor Mr. Robner was done in.

First off, type "EXAMINE RUG" (or just "LOOK RUG"). You will find some mud spots which is your first clue. Now, "GET THE CUP, PAD, CALENDAR AND PENCIL" and "RUB PENCIL ON PAD" and then "TURN PAGE OF CALENDAR." Aha! Perhaps a clue as to the motive? Let's see if we can substantiate the method a little more...that mud on the rug was very interesting. Type "OPEN BALCONY DOOR" and go [N] onto the balcony. Check out the railing by typing "EXAMINE RAILING" and you will see some scratches, lending credence to the theory that perhaps the murderer climbed up the balcony from the ground below where he (or she) got mud on his shoes. Let's have a look below and check for some indication that the murderer was indeed below the balcony.

To leave the balcony, go [S,S,E,E,E,E,D,D,W,S]. Type "OPEN DOOR" and go [S] back to the front door. Now go [E,E,SE] to the shed where you will see a ladder. Type "EXAMINE LADDER."

Hmmm! This ladder- and-balcony theory is looking good! Let's see if we can prove the ladder was below the balcony. This will have to wait a while, though, because it's getting late in the morning and we have to do some more checking in the house before the reading of the will takes place. And besides that, we need to talk to Mr. McNabb and he doesn't seem to be in the mood right now.

Go back to the house by heading [N,S,N] and head back upstairs with [N,N,E,U,U]. Let's see what else we can find upstairs. Go [S,S] into Dunbar's bathroom. Type "OPEN CABINET" and "EXAMINE LOBLO." Aha, again!
Now we go back downstairs and see if we can find Mr. McNabb to see if he knows anything about a ladder under the balcony. Go [N,N,D,D,W,S,S]. Let's take a break for a while. Type "WAIT UNTIL 11:30."

And now for Mr. McNabb. Let's try the garden path first with [E,NE,E,W]. If McNabb is not around, just wait for a while or snoop around the area and he will soon show up. Deadline is very unpredictable when it comes to the various characters moving around the scenario. Once you spot McNabb, go to him and say "HEY MCNABB" followed by "WHAT IS WRONG." He will tell you about some holes he found in his garden so, naturally, you say "SHOW ME THE HOLES." He will take off and you "FOLLOW HIM." When he stops, type "EXAMINE HOLES." Eureka! The ladder was here and the depth of the holes proves somebody climbed it up to the balcony!
To make sure we cover every angle, type "EXAMINE GROUND" and "DIG AROUND HOLES." Hmmm...wonder what this could be about? To find out, type "ANALYZE FRAGMENT FOR LOBLO." Oops, it's later than we thought!

Back to the house for the reading of the will. Go [N,SW,SE,E] to the house and [N,N,W] into the living room. Now just "WAIT" for the will to be read.

After the will is read, you decide to see if you can roust some of those present into giving you some clues as to the guilty party and, perhaps, the motive for the crime. Let's start with George. Type "SHOW GEORGE THE CALENDAR." He will get very nervous and start heading out of the room.
Type "FOLLOW HIM" until he finally goes to his room. He will keep telling you to leave him alone, but just keep following him until he enters his room. At this point, you decide to see if George knows more then he's telling. You aren't going to get anything from him here, so let's go to the balcony and wait to see if he does anything. Go [W,N,N] to the balcony and type "WAIT 10 MINUTES." Voila! Here he comes! Wait until he goes behind the bookshelf and then type "WAIT 4 MINUTES" to give him time to really get his hands into the cookie jar.

When your 4 minutes are up, go [S], "EXAMINE BOOKSHELF," "PRESS BUTTON," and go [E]. Ha! Caught him red-handed!! Type "GET WILL," "LOOK SAFE," "GET PAPERS," and "READ PAPERS." Things are beginning to look up! Let's see if we can substantiate some of this stuff. Go back to the living room with [W,S,E,E,E,E,D,D,W,W]. My, isn't this cozy! Type "HEY BAXTER," "WHAT ABOUT FOCUS." You know he's lying so you "SHOW PAPERS TO BAXTER." Ah, that's better! Now for some clever psycho-detective work. Type "SHOW LAB REPORT TO DUNBAR" and "SHOW LAB REPORT TO BAXTER." Whip around and "ACCUSE DUNBAR." Hmmm...a tad nervous, isn't she? Perhaps we should go off and wait to see what develops. Go [E] to leave the room and "WAIT FOR DUNBAR." Just as we suspected! When she passes you, type "FOLLOW HER." Once outside the house, she will drop a ticket. Type "GET TICKET" and "READ TICKET." WOW! This is getting good! Type "SHOW TICKET TO DUNBAR." You know you've got her on the run now so head off to the shed to wait and see what develops. Go [E,E,SE] and "WAIT FOR BAXTER." When they both show up, "SHOW TICKET TO BAXTER" and "ARREST BAXTER AND DUNBAR." You didn't believe them for a minute, did you?

Due to the dynamic nature of Deadline, there are several ways to end up accusing Baxter and Dunbar of the murder. There are also more puzzles to solve, but this is all that is necessary to put together an air-tight case against them. If you have other methods of solving Deadline, let us know!

ROUND() FUNCTION FOR PRO-MC

by J.F.R. "Frank" Slinkman

One function which is found in some implementations of the C programming language, but which is not included in the Pro-MC package from Misosys, Inc., is a function to round floating point numbers to their nearest integer value.

Of course, numbers can be rounded in code in C just as they can in BASIC, through code something like:

```
double round( value )
double value;
{  return floor( fabs( value ) + 0.5 ) * dsgn( value );
   }
```

While the above does the job, it calls several subroutines built into the libraries (e.g., DSGN to get the sign, FABS to get the absolute value, @ADDD to add 0.5 to the absolute value, FLOOR to calculate an integer, @MULTD to multiply, not to mention several hidden calls to move data to and from various RAM locations), which make the operation fairly slow.

Obviously, both convenience and speed are valid reasons for adding a round() function to the Pro-MC math library.

But, before we can start playing with double precision floating point numbers, we first have to learn how these numbers work, and how their values are stored (which is exactly the same in Pro-MC and TRS-80 BASIC).

Double precision numbers are stored in 8 bytes. The first seven bytes hold the mantissa and the sign of the value, in order of least-significant to most significant bytes. The 8th byte carries the exponent, which is the power of two by which the mantissa in the first seven bytes must be multiplied.

This exponent is stored "base 128," which means 127 means -1; 128 means 0, 129 means +1, etc. This gives us an exponent range from -127 to +127. By convention, an exponent of zero, instead of meaning -128, means the entire stored value is zero.

The first 7 bytes are always considered to hold a value of at least 0.5 but less than 1.0. This puts an implied radix point (like a "decimal" point, but this is a binary system, not a decimal system) to the left of the most significant bit. Also, the most significant bit under this scheme will always be "1;" so it, too, can be implied. Thus, bit 7 of the 7th byte is free to be used to hold the sign of the number, under a convention where "1" = negative and "0" = positive.

Thus:

+0.5 is stored as 00 00 00 00 00 00 00 80 (hex);
- 0.5 is stored as 00 00 00 00 00 00 80 80
+3.5 is stored as 00 00 00 00 00 00 60 82
- 7.0 is stored as 00 00 00 00 00 00 E0 83; etc.

To fully understand how values are stored, let's break down the "+3.5" example. First arrange all the significant bits in correct order. To do this, because in this example there are no set bits in the first 6 bytes, we need only take the 7th (the most significant) byte (60H), and express it in binary with it's implied radix point:

60H -> 0.01100000 (binary)

Now set the implied high order bit to get:

0.11100000 (binary).

In this binary system, the first bit position to the right of the radix has a place value of 2^-1 (0.5); the second has a place value of 2^-2 (0.25), and so on through descending powers of 2.

Thus the first bit in this example means 1/2. The 2nd bit means 1/4; and the 3rd means 1/8. When we add all that together, we get 7/8ths, or 0.875.

Now we take the exponent (82H = 130) and subtract 128 from it to get 2.

The total value, then, is (0.875 * 2^2) = (0.875 * 4) = 3.5.

Now, if you think about it, you'll realize that the integer portion of the number (3 in this case) has to have exactly as many bits as the value of the exponent (2 in this case). Therefore, since there are 56 bits available to express both the integer and fractional portions, there must be 56 minus the

exponent value bits available to express the fraction (56 - 2 = 54, in this case). Remember these facts -- we will make use of them later.

So, to round the number, all we have to do is look at the first bit after the bits which express the integer portion. If it's zero, we don't have to round up. If it's one, we must round up. In this case, the 3rd bit is one; so we would have to round up. This would be done as follows:

```
 0.11100000B =  (0.875 * 2^2) = (0.875 * 4) = 3.5
+0.00100000B =  (0.125 * 2^2) = (0.125 * 4) = 0.5
 ------------
 1.00000000B =  (1.000 * 2^2) = (1.000 * 4) = 4.0
```

But now we have to do some juggling, since our rounded result no longer fits the rule that all binary values must be in the range 0.5 <= value < 1.0.

To fix this, we have to shift the mantissa right one bit (the same as dividing by 2), and increment the exponent (the same as multiplying by 2). After these operations, the result becomes:

```
0.10000000B =  (0.500 * 2^3) = (0.500 * 8) = 4.0.
```

Putting this back in double precision storage format, we get:

00 00 00 00 00 00 80 83.

Now we reset the implied bit, to get:

00 00 00 00 00 00 00 83.

And we're done.

Of course, the above example is simple, because all the bits to the right of the ones we looked at were zeros. Suppose, for example, we had to round a numbers like 9.335 and -16.728.

These two numbers are stored (in hex) as follows:

9.335 as 5B 8F C2 F5 28 5C 15 84; and

-16.728 as 6C E7 FB A9 F1 D2 85 85;

While 9.0 and -17.0 (the rounded values to be derived from the above) would be expressed as:

9.0 as 00 00 00 00 00 00 10 84; and

-17.0 as 00 00 00 00 00 00 88 85.

Obviously, in addition to keeping track of the sign, possibly adding, shifting, and adjusting exponents, a round() function will also have to reset all bits in the fractional part of the number to zero.

O.K. Now that we know what we have to do, let's look at one way to go about actually doing it. It's now time to look at the listing of ROUND/ASM.

Since this is going to be a function to be added to the Pro-MC libraries, we have to be aware of the way double precision values are passed to and from functions.

An argument is passed TO the function on the stack; which means it starts at SP+2, since the return address will be at SP+0. Values are returned FROM the function in a special 8-byte buffer. Values are loaded into this buffer by pointing HL to the 1st byte of the value, and calling a Pro-MC routine named @GINTO. Using @GINTO means we don't have to know where the buffer is -- we just let the system take care of that for us.

Also, because "register" variables can be held in the index registers IX and IY, we can only use those registers if we make sure they hold the same values they did upon entry into our routine.

Thus, the first instruction at ROUND is to PUSH IX, saving it so it can be restored for a clean exit. This means the argument (the double precision value passed to the function) is no longer at SP+2, but at SP+4.

So we utilize the MCMACS/ASM macro "$HS" to point HL to the argument. All "$HS 4" does is load HL with 4, then add SP to it, causing HL to point to the stacked argument.

Now we copy this pointer to IX, and pick up the 7th byte (@ IX+6), which contains the sign bit, which we isolate and store in the C register.

Now the exponent in the 8th byte is picked up and examined. If its value is < 128 (i.e., < 0), then the absolute value of the argument is < 0.5. Because the rounded result will be zero; we just set the exponent to zero and return the altered argument.

If the exponent is equal to 128 (i.e., 0), then 0.5 <= absolute value < 1, which means we round the absolute value to 1.0, restore the sign bit, and exit.

If the exponent is >= 184 (128 + 56), then all 56 bits in the first 7 bytes are integer bits (i.e., the whole value is so large it's already an integer); so there is no need to do anything but return the original argument.

However, if the exponent is in the range 129 (2^1) to 183 (2^55), the program actually has to do some work. First at the NEG instruction at RND030+2, we calculate the number of bits contained in the fractional portion of the number.

(If you'll recall, we determined above that the integer portion will have the same number of bits as the value of the exponent.)

Next, we set the implied bit to fully express the value, and generate a pair of masks in registers D and E. The mask in the D register maps the location of the most significant bit of the fractional portion of the number, while the mask in E maps the integer bits which must be preserved. You'll note in the comments above label RND050 that the single set bit in the D mask is one bit to the right of the rightmost set bit in the E mask.

At RND050, the number of full bytes to be cleared is calculated, and from that is derived the number of bytes which are either wholly or partially used by the integer portion of the value, which number is stored in B.

In the loop starting at RND060, every byte which can be safely nulled out is set to 00H. Note this loop tests for 9 bits, rather than the normal 8 bits per byte, since the possibility exists that the break between integer bits and fraction bits could occur at an even byte boundary.

If the break is not at a byte boundary, the program goes to RND080.

If the break is at a byte boundary, the value in D is added to the byte pointed to by HL to determine whether or not rounding up is needed, and then the byte at (HL) is set to 00H, completing the resetting of all fraction bits.

If the C flag is reset, then the fraction was < 0.5; so we're essentially finished. All we have to do is go to RND130 to restore the original sign bit and return the altered argument as the function return value.

If the C flag is set, then we must bump the counter in B to compensate for the fact HL points to the last fraction byte and not the first integer byte, whereafter control goes to RND090.

At RND080, the break between integer and fraction occurs in the same byte; so the value in D is added to the byte, and then the fractional bits are set to zero through application of the mask in the E register. Again, if the C flag is reset, no round up is needed; so control passes to RND130, where the sign bit is restored and the altered argument returned.

The code at RND090 does the rounding up by adding the carry flag from each byte to the next throughout the entire integer portion. If, at the end of this process, the C flag is reset, then the value in the first seven bytes is still < 1.0; so we're essentially finished.

However, if the C flag is set, then the value is 1.0 or greater; so it is adjusted by incrementing the exponent, and shifting the mantissa one bit right. The reasons behind this were explained earlier.

Finally, at RND130, the implied bit is reset, and the sign bit copied to that location. The pointer to the LSB of the altered argument is copied from IX to HL, the original IX popped off the stack, and the return value copied to the Pro-MC system buffer via a call to @GINTO.

The operation is now complete; so the routine returns to its caller.

To determine the efficiency of this routine, I wrote a short program which recorded the time it took to make 30,000 calls to round up and store the value 6.666667. On my XLR8er equipped Model 4D, it accomplished this in about 10.5 seconds.

By contrast, the same test applied to the "normal" method of rounding (see example code above) took about 75 seconds.

Because this routine does not use any of the other functions in MATH/REL, including it in your Pro-MC libraries is easy. Basically, you have two equally good options:

1. If you have the Misosys, Inc., utility MLIB, you can load MATH/REL, and add the module, giving it the name ROUND. I put it between the INTRND and CEIL modules because it seems like it "belonged" there, but the location really doesn't matter;

2. You can add the module to MATH/REL by issuing the LS-DOS command:

APPEND MATH/REL ROUND/REL (STRIP)

Once you have added the ROUND module to MATH/REL, you should edit MATH.H to include "round()" (excluding the quotes) to the list of declared extern doubles. This eliminates the need to remember to declare it in every program you write.

Also, in programming, remember that round()

expects a double argument. Normally, floats are automatically converted to doubles before being passed as arguments to a function. However, if you use the Pro-MC compiler "+f" option, this is no longer true, and any float to be rounded must be explicitly cast to double before being passed to the function.

Using the Pro-MC documentation format, the documentation for round() follows:

round(MATH) round(MATH)

```
#include <math.h>
double round( darg );
double darg;

darg    is the double whose
        rounded integer value is
        to be determined.
```

This function is used to obtain the rounded integer portion of a double.

Description
This function will obtain the rounded integer portion of its double precision argument and return a double precision result.

Return Code
There is no special return code other than the double precision result.

Warning
Round() expects a double precision argument. Passing another type of argument will produce erroneous and unpredictable results. When using the compiler "+f" switch, float arguments must be explicitly cast to double before being passed to round().

Example

```
#include <stdio.h>
#include <math.h>
char   inbuf[81];
double  d1, d2;
main()
{  puts( "round: enter your number: EOF to exit" );
   while( gets( inbuf ) )
   {  d2 = round( d1 = atod( inbuf ) );
      printf( "% f rounds to % f\n", d1, d2 );
   }
}
round: enter your number: EOF to exit
3.333      | 3.333000 rounds to  3.000000
-6.6667    |-6.666700 rounds to -7.000000
0.4825     | 0.482500 rounds to  0.000000
```

ROUND/ASM

```
; ROUND/ASM
; Author: J.F.R. "Frank" Slinkman
;        1511 Old Compton Road, Richmond, VA 23233
;        CompuServe 72411,650
; Date:  4-Oct-93
; Copyright (c)1993, J.F.R. "Frank" Slinkman
; All rights reserved.  Individuals are hereby granted per-
; mission to make private, non-commercial use of this code.
;
;Register usage:
;        IX, HL  pointers
;        B       counter
;        C       sign storage (80H = minus, 00H = plus)
;        D       fraction msb bit mask (rounding bit)
;        E       integer bit mask (bits to save)
;
;In comments, LSB & MSB refer to bytes, lsb & msb
; refer to bits
;double round( arg )
;double arg;
;   returns value of "arg" rounded to nearest whole number
;
*GET    MCMACS
;
        PUBLICROUND
;
ROUND:
        PUSH    IX              ;save poss reg variable
        $HS     4               ;-> LSB of arg
        PUSH    HL
        POP     IX
        LD      A,(IX+6)
        AND     80H             ;isolate sign bit
        LD      C,A             ;and store
        LD      A,(IX+7)        ;p/u exponent
        CP      128             ;arg >= 0.5?
        JR      NC,RND010       ;go if so
        LD      (IX+7),0        ;else if < 0.5,
                                ;round to zero
        JR      RND140          ;go to ret modified arg
RND010:
        JR      NZ,RND030       ;go if exp >= 129
                                ; (i.e. arg >= 1.0)
        XOR     A               ;else 0.5 <= arg < 1.0;
                                ;so return 1.0
        LD      B,7
RND020:
        LD      (HL),A          ;nul out 56 fractional
                                ;bits (7 bytes)
        INC     HL
        DJNZ    RND020
        INC     (HL)            ;bump exponent to 129
        JR      RND130          ;go to return 1.0
RND030:
        SUB     128+56          ;is arg so large it's
```

```
                                ;already an int?
        JR      NC,RND140       ;return entry value if so
        NEG                     ;calc # of bits in
                                ;fractional portion
        LD      B,A             ;shift counter
        SET     7,(IX+6)        ;set implied bit
        LD      DE,128<8!255    ;D = 80H, E = 0FFH
        LD      A,7
        AND     B               ;A = # of bits to shift
        JR      Z,RND050        ;go if shifts evenly
                                ;divisible by 8
RND040:
        RLC     D               ;shift rounding bit
        SLA     E               ;reset rightmost bit
        DEC     A
        JR      NZ,RND040
;
;shifts  D              E
; 0      10000000       11111111   0 is special case
(mask different bytes)
; 1      00000001       11111110   all others mask
; 2      00000010       11111100   same byte
; 3      00000100       11111000
; 4      00001000       11110000
; 5      00010000       11100000
; 6      00100000       11000000
; 7      01000000       10000000
;
RND050:
        LD      A,B             ;# of bits to clear in A
        PUSH    AF
        SRL     A               ;divide # of bits by 8
                                ;bits/byte
        SRL     A               ;quotient range: 0 to 6
        SRL     A
        SUB     7               ;calc # whole or partial
                                ;bytes which will
        NEG                     ;contain integer bits
                                ; (range 1 to 7)
        LD      B,A
        POP     AF
RND060:
        CP      9               ;can this byte be nulled
                                ;out?
        JR      C,RND070        ;go if not, or not yet
        LD      (HL),0          ; else null out the byte
        INC     HL              ;-> next byte
        SUB     8               ;reduce by one byte's
                                ;worth of bits
        JR      RND060
RND070:
        CP      8               ;int lsb & fraction msb
                                ;in different bytes?
        JR      C,RND080        ;go if not
        LD      A,D             ;p/u roundup value
                                ;(must be 80H if here)
        ADD     A,(HL)          ;add it
        LD      (HL),0          ;now null out the last
                                ;nullable byte
        JR      NC,RND130       ;go if no round up
                                ;needed
        INC     B               ;compensate for byte
                                ;boundary situation
        JR      RND090
RND080:                         ;here if int lsb and
                                ;fraction msb in same
                                ;byte
        LD      A,D             ;p/u value to add to
                                ;round up
        ADD     A,(HL)          ;add it
        PUSH    AF              ;save C flag
        AND     E               ;preserve only int bits
        LD      (HL),A          ;this resets fract bits
        POP     AF              ;restore C flag
        JR      NC,RND130       ;go if no round up
                                ;needed
RND090:
        LD      D,0             ;for efficiency
        INC     HL              ;-> next byte
        DJNZ    RND100          ;go if not at last byte
                                ;(adjusts B)
        JR      RND110          ;if at last byte,
                                ;HL -> exponent
RND100:
        LD      A,D
        ADC     A,(HL)          ;perform carry from
                                ;previous byte
        LD      (HL),A          ;replace value
        INC     HL              ;when done,
                                ;HL will -> exponent
        DJNZ    RND100
RND110:
        JR      NC,RND130       ;go if rounded value
                                ;still < 1.0
        INC     (HL)            ;else bump exponent
                                ; (multiply by 2)
        LD      B,7             ;& divide fraction by 2
RND120:
        DEC     HL
        RR      (HL)
        DJNZ    RND120
RND130:
        LD      A,(IX+6)        ;p/u MSB of mantissa
        RES     7,A             ;reset implied bit
        OR      C               ;apply sign mask
        LD      (IX+6),A        ;replace
RND140:
        PUSH    IX
        POP     HL              ;-> LSB of modified arg
        POP     IX              ;restore poss reg var
                                ; (clears stack)
        CALL    @GINTO##        ;get arg into MC system
                                ; buffer to return
        RET
        END
```

TIRED OF SLOPPY DISK LABELS?
TIRED OF NOT KNOWING WHAT'S ON YOUR DISK?

YOU NEED "DL"

"DL" will automatically read your TRSDOS6/LDOS compatible disk and then print a neat label, listing the visible files (maximum 16). You may use the 'change' feature to select the filenames to print. You may even change the diskname and diskdate. "DL" is written in 100% Z-80 machine code for efficiency and speed.

"DL" is available for TRS-80 Model 4/4P/4D using TRSDOS 6.2/LS-DOS 6.3.0 & 6.3.1 with and Epson compatible or DMP series printer.

"DL" for Model 4 only $9.95

TRSTimes magazine - Dept. "DL"
5721 Topanga Canyon Blvd., Suite 4
Woodland Hills, CA 91367

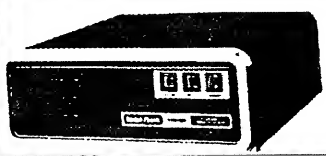
HARD DRIVES FOR SALE

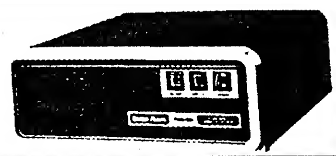
Genuine Radio Shack Drive Boxes with Xontroller, Power Supply, and Cables. Formatted for TRS 6.3, Installation JCL Included. Hardware write protect operational. Documentation and new copy of MISOSYS RSHARD5/6 Included. 90 day warranty.

5 Meg $175 | **10 Meg $225** | **15 Meg $275** | **35 Meg $445**

Shipping cost add to all prices

Roy T. Beck
2153 Cedarhurst Dr.
Los Angeles, CA 90027
(213) 664-5059





PROGRAMMING TIDBITS

Copyright 1993 by Chris Fara (Microdex Corp)

This article is a reprint from our last issue where we, unfortunately, succeeded in completely mangling Chris Fara's fine submission to the point that it became unreadable. In reviewing the problem, I found that some text was missing, while other text was printed out of sequence — and more than once. How did this happen? Was it a glitched disk? No, I checked that. The disk is good. Could it be the software? Yes, but in all honesty I must say "probably not". Could it be the hardware? Yes, but again, "probably not". So, what is the problem? I haven't a clue — but the bottom line is that I should have caught the mess before it went to print. My sincere apologies to Chris and all the readers.

Ed.

SCREEN ROBOT III

Why couldn't a computer automatically write programs for us? It has been attempted, but so far with more hype than meat. Some day, maybe. Meanwhile, instead of wasting money on dubious promises, we might try something simple yet useful on our own. For instance, one tedious programming chore is the layout of screens: menus, help screens, data entry forms, and similar displays. So here is an idea that might help a bit. It will instantly produce a file with a screen display subroutine that later can be simply MERGEd into any BASIC program. Just so you Mod-III people know we care, we'll start with a Mod-III version (Mod-4 later).

SCREEN EDITOR

First we need some sort of "editor" to lay out the screen. We'll sketch here only its simplest skeleton. If you like the idea, you can make it more fancy later.

```
10 clear 5000: cls: defint a-z
11 x=0:y=0:xx=64:yy=16              'video size
12 d$=chr$(9)+chr$(8)+chr$(10)+chr$(91)
13 d$=d$+chr$(13): q$=chr$(34)
20 print @ y*xx+x, chr$(14);        'cursor
30 k$=inkey$: if k$="" then 30
31 print chr$(15);                  'cursor off
32 on instr(d$,k$) goto 41,42,43,44,50
33 if k$>=" " then print k$;:else 20
41 x=x - (x+1 < xx): goto 20        'right
42 x=x + (x > 0)   : goto 20        'left
43 y=y - (y+1 < yy): goto 20        'down
44 y=y + (y > 0)   : goto 20        'up
50 '-----save
```

After the usual start-up chores (clear string space, etc.) define variables: X and Y are cursor coordinates, XX and YY are screen limits, D$ holds ASCII codes for arrows: right, left, down and up, plus the ENTER key, and Q$ is a quotation mark for writing strings to file.

In line 20 calculate the screen position from X and Y, and turn on the cursor. In line 30 wait for a key, and when one is hit, turn off the cursor in case we need to move it. Then try INSTR to see if an arrow or ENTER was pressed: if so, then branch to one of the arrow handling lines, or to the "save" routine in line 50 to be discussed in a moment. If none of the codes match then the program "falls thru" to line 33: if we have a displayable character (blank space or higher) then display it. Otherwise go back for another key.

After displaying the character (semi-colon prevents cursor from dropping to the next screen row) our program goes to next line which also happens to be handling the right arrow. Even though PRINT already moved the cursor to the right, we need to calculate the new X-coordinate here: it will increase by 1, but only if the cursor is not yet at the right edge of the screen. In BASIC a comparison such as.....

$$A < B$$

generates -1 if true, 0 if false. So, if adding one will result in a column number X lower than XX (ie. 63 or less) then the cursor will advance by 1 step, because.....

$$X - (X+1 < XX) = X - (-1) = X + 1$$

Otherwise the comparison is "false" and leaves the cursor position unchanged...

X - (X+1 < XX) = X - (0) = X

In a similar way coordinates are calculated in lines 42-44 for other arrows. Then go back to line 20, turn on the cursor at the calculated position and wait for a key.

Even with this rudimentary "editor" we can now easily scribble all over the screen: use arrows to move around, type something, change, erase with SPACE bar, and so on. More code can be added to copy and move text, etc, but that's not the point here.

PROGRAM WRITER

When the screen looks right, press ENTER to start generating a BASIC program file. Three steps are involved: get the screen rows from memory, trim all blanks from left and right sides of each row, and construct BASIC command lines for output to file. The program lines in that file will have this "syntax".....

PRINT @ position,"text";

The semicolon at the end comes in handy when something is displayed in the bottom line: it prevents scrolling of text (as long as no character is in the rightmost corner of the screen). Let's try it.

```
50 '------save screen
51 open "O", 1, "SCREEN/SUB"
52 print #1, "10000 CLS"
53 a$ = string$(64,32)
54 for y=0 to yy-1              'get screen rows
55 poke varptr(a$)+1, y*64 and 255
56 poke varptr(a$)+2, &H3C + fix(y/4)
```

We name the file SCREEN/SUB but you can change it to anything you want. Similarly, the lines of our subroutine will be 10000 and up, but could be anything convenient. The first line will simply clear the screen. If for some reason you don't want to clear the screen every time the subroutine is called, then omit the CLS, but make sure that we have a line 10000 (or whatever number you choose), perhaps with some REMark, so that the subroutine can always be called with the same standard line number. As will be seen later, the remaining files of the subroutine may not always be consecutive.

Next, create a string A$ with the same length as the screen width (64 blanks) and start a FOR loop to scan all the screen rows. The next two lines change the "variable pointer" of string A$ so that it now points to the screen memory. That memory in Mod-III starts at hex '3C00 where the image of the top screen row #0 is stored. The second row #1 is stored at hex'3C00 plus 64, and so on. Our POKEs calculate the low and high byte of those addresses for each Y=row number. When Y*64 gets to be 256 or more (after row #3), the AND will mask out the excess of 256. Thus for the row #4 we'll poke 0 (same as for the top row), for the row #5 we'll poke 64, and so on. At the same time the high byte poke will be incremented for every fourth row whenever we pass the 256-byte boundary. If you're not comfortable with such "binary" capers then just take my word for it: it is so.

Anyway, after the two POKEs the picture of a screen row sits in the string A$. But we don't want to clutter our subroutine file with useless blank spaces, so the next task is to trim all leading and trailing blanks. And, if in the process we discover that an entire row is blank, then we'll ignore it altogether.

```
60 j=xx+1                    'trim string
61 for x=1 to xx             'leading blanks
62 if mid$(a$,x,1)>" "then i=x:x=j:j=i
63 next: if j>xx then 80
65 for x=xx to 1 step -1     'trailing
66 if mid$(a$,x,1)>" "then k=x:x=1
67 next: k=k-j+1
```

First scan the string from the beginning. When a non-blank character is found, then the counter variable X is swapped with J. Therefore now J contains the position X in the string where the first non-blank was found (the column on the screen where the text begins) while X equals XX+1 and thus forces NEXT in line 63 to terminate the scan.

Now, if all characters were blank then after NEXT the value of J is still XX+1 (it was pre-set in line 60 and never got swapped), J>XX is true, and we skip to line 80 to try the next row (see next program segment below). If the string is not blank then lines 65-67 scan backwards for trailing blanks, and calculate the length K of that portion of A$ which will remain after discarding any leading and trailing blanks.

You will notice that, when the screen is mostly blank, the "trimming" takes a couple of seconds (hundreds of MID$ comparisons are made). But this is time well spent, because it eliminates tons of useless blanks from the resulting subroutine file.

The next, final segment of our "robot" program

writes a line to the disk. For educational purposes the PRINT#1 command has been split here into three pieces, but it could be just as well written in one line. In any case on the disk it will create one BASIC command line.

```
70 print #1,10000+1+y;      'write
71 print #1, "PRINT @" y*xx+j-1 "," ;
72 print #1, q$  mid$(a$,j,k)  q$ ";"
80 next                     'go get next row
81 print #1,"10029 RETURN"
82 close: end
```

First, a line number is constructed by adding the current row number (0-15) to a "base" number (in our case 10000+1, because the "header" line in our subroutine was at 10000). Then the PRINT@ keyword and screen position is added. The position is calculated from row Y and column J where the non-blank part of the screen row begins. We must subtract 1 because J=1 if the string starts at the leftmost column, but in PRINT@ this must be column 0, etc. Finally, write the text of A$ surrounded by quotes O$, plus semicolon to stop the cursor from dropping down.

Note that the subroutine lines generated by our "robot" are numbered sequentially, but not necessarily consecutively: for example if the top screen row (row 0) is blank then the subroutine will not have line 10001, and so on. That's why it is a good idea to start the subroutine with some "neutral" line that will be always present, as we have done with our line number 10000.

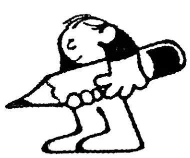
After all rows are done, write the RETURN command and close the file. The file looks just like any BASIC program file saved with the "A" (ASCII) option and can be now included in any program.....

```
MERGE "SCREEN/SUB"
```

Then, to display the screen.....

```
GOSUB 10000
```

Pretty sleek for being so simple, isn't it? Next time we'll look at Mod-4. By the way, both Mod-III and Mod-4 versions of the "screen robot" are available, along with other "goodies" on the... "Goodies" disk from Microdex ($12.95+SH; address and phone number see Microdex ad elsewhere in this issue).



MORE PROGRAMMING TIDBITS

Copyright 1993 by Chris Fara (Microdex Corp)

SCREEN ROBOT 4

Last time we've explored a simple BASIC program for Model III, to automate writing of screen display routines. A simple screen editor was used to lay out the display, and another routine translated the display into BASIC program lines to be MERGEd into other programs. In Mod-4 the idea works the same way, except for a minor obstacle: video memory is not directly accessible. The only way to get at that memory is via machine programs.

Fortunately there is LS-DOS 6.3. The BASIC included with it has a new special "user" call command, USR11. This command provides direct access from BASIC to DOS SuperVisor Calls (SVC). It works like this: DIMension an integer array, for example.....

```
DIM Z% (5)
```

then load element(0) with the number of the desired SVC, load other elements of the array with values for Z80 registers needed by the SVC, and call SVC like this.....

```
Z% = USR11 (VARPTR(Z%(0)))
```

We are interested in SVC 15 to get a row from screen into BASIC. This SVC expects the following Z80 registers.....

H	row number 0-23
DE	address of our storage buffer
B	9 (function number)
C	1 (copy from screen to buffer)
	Other registers are irrelevant.

As with Robot-III, we'll get the screen row into a string variable, except it will have 80 characters. But there is another twist to that. Strings normally are stored in high memory. When SVC 15 is executed, that high memory is temporarily swapped with video memory and some strings may become inaccessible. So we must force our string into a lower part of memory. One way to do that is to write a "literal" string right near the beginning of the program, and never assign to it any new value (which would move it into string area in high memory and thus defeat its purpose).

MOD-4 SCREEN EDITOR

With this in mind, let's outline the first part of our program.

```
10 cls: defint a-z
11 x=0:y=0:xx=80:yy=24  'video size
12 d$=chr$(9)+chr$(8)+chr$(10)+chr$(11)
13 d$=d$+chr$(13): q$=chr$(34)
15 z=0: dim z(5)             'array for USR11
16 a$="Here type exactly 80 characters"
20 print @ y*80+x,;
30 k$=inkey$: if k$="" then 30
32 on instr(d$,k$) goto 41,42,43,44,50
33 if k$>=" " then print k$;: else 20
41 x=x - (x+1 < xx): goto 20          'right
42 x=x + (x > 0)   : goto 20  'left
43 y=y - (y+1 < yy): goto 20          'down
44 y=y + (y > 0)   : goto 20  'up
50 '-----save
```

Predefine a variable Z and array Z(5) for USR11. The string A$ will be used for transfer of video rows. Type 80 blanks, or anything else, as long as there are exactly 80 characters between the quotes. The rest is like in the Mod-III "robot", except for different screen limits and different code for "up" arrow. Also, there is no need to turn the cursor on and off. Again, this is a very rudimentary editor, and if you like the whole idea, you'll probably want to improve it later.

MOD-4 PROGRAM WRITER

The main difference from Mod-III robot is the way we get the screen row with the "special" USR11 command (remember, it works only with DOS 6.3 BASIC).

```
50 '-----save screen subroutine
51 open "O", 1, "SCREEN/SUB"
52 print #1, "10000 CLS"
52 for y=0 to yy-1              'get screen rows
53 z(0)=15                      'SVC number 15
54 z(1)=256*Y                   'H=row number
55 z(2)=peek (varptr(a$)+1) +
     256*peek (varptr(a$)+2) - 65536
56 z(3)=256*9+1                 'B=9  C=1
57 z=USR11 (varptr(z(0)))
```

Line 55 calculates the address of our A$ string (write it as one single line). This could be also pre-calculated once near the beginning of the program, because that string by definition will not move. But the address must be inserted into Z(2) before every call, because SVC may change it.

The result at this point is that the picture of a screen row sits in the string A$. Run the program, type something in the bottom row of your screen layout, and then LIST the program: you'll see a bit of magic. The bottom screen row will now sit in line 16 of your very own program listing, even though you didn't put it there!

The rest is the same as in the Mod-III version of the "robot". In line 62 you may use the convenient Mod-4 command SWAP.....

```
60 j=xx+1                       '-----trim string
61 for x=1 to xx                '---leading blanks
62 if mid$(a$,x,1)>" "then swap j,x
63 next: if j>xx then 80
65 for x=xx to 1 step -1        'trailing
66 if mid$(a$,x,1)>" "then k=x:x=1
67 next: k=k-j+1
70 print #1,10000+1+y;          'write
71 print #1, "PRINT@" y*xx+j-1 "," ;
72 print #1, q$  mid$(a$,j,k)  q$ ";"
80 next                         'go get next row
81 print #1,"10029 RETURN"
82 close: end
```

Again, MERGE the subroutine into any program and then GOSUB 10000 to display the screen.

MORE ABOUT USR11

If you don't have LS-DOS 6.3 yet, then get it. Besides extending the date range into the 21-st century, it is altogether a better DOS for MOD-4 ("Mod-4 by Chris" manual fully updated for 6.3.1 is now also available from Microdex, see ad elsewhere in this issue).

The direct access to DOS SuperVisor Calls, via USR11 command in the BASIC included with DOS 6.3 upgrade, is so handy that it may be useful to review its details here. As noted earlier, USR11 needs an integer array, such as.....

```
DIM Z% (5
```

The name of the array may be one or two characters, but no more, else an error occurs. Also, DEF USR11 is not needed and not even allowed (causes "syntax error").

In the first element put the number of the desired SVC, and in the remaining elements put the values for Z80 registers, expected by the SVC. The values go into the array like this.....

```
Z% (1) = value for register HL
Z% (2) = value for DE
Z% (3) = for BC
Z% (4) = for IX
Z% (5) = for IY
```

The array must always have the subscripts 0-5 and the SVC number in element (0), but in elements (1) through (5) only those values need to be specified which are actually required by a given SVC.

Finally execute the SVC like this.....

```
Z% = USR11 ( varptr( Z% (0) ) )
```

Upon return to BASIC the element Z%(0) contains the value of register pair AF (F are the flags), and other elements contain the values of the other registers as may or may not have been modified by the SVC.

Say, for example, you want to change the cursor shape from BASIC. In Mod-III this is easy with a simple POKE, for instance.....

```
POKE 16419,95          'Mod-III          only!
```

to change the standard block cursor to an underscore. In Mod-4 BASIC we could of course use the SYSTEM command. For example.....

SYSTEM "SYSTEM (BLINK=143)"

will change the cursor to a graphic block. But that's a lot of disk grinding for a tiny cursor. The same thing can be done instantly and silently via USR11.....

```
Z%(0) = 15                  'video SVC 15
Z%(3) = 256*8 + 143         'B=8, C=cursor
Z% = USR11 (varptr(Z%(0)))
```

Or perhaps your BASIC program needs to know the amount of free disk space before saving a big file. Elementary, my dear Watson. Try drive 1....

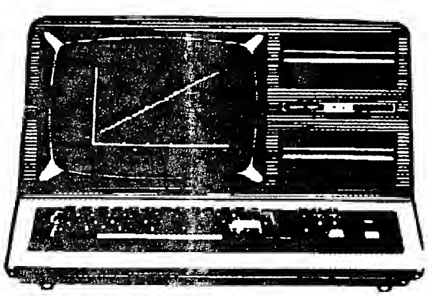
```
A$ = string$(20,32)         'buffer for SVC
Z%(0) = 34                  'directory SVC
Z%(1) = peek(varptr(A$)+1) +
256*peek(varptr(A$)+2) - 65536
Z%(3) = 256*4 + 1           'B=4, C=drive #
USR11 (varptr(Z%(0)))
FREE% = cvi (right$ (A$,2))
```

Now the variable FREE% contains the amount of free disk space in K-bytes.

And so on. Each SVC expects and returns various values in registers (and sometimes in "buffers") as described, for example, in "Mod-4 by Chris". Since each element of an integer array consists of 2 bytes, we multiply the value for the "high" register times 256 and add the value for the "low" register. In the above example SVC 34 needs "function number" 4 in register B and drive number in register C. So the expression.....

```
Z%(3)   =   256   *   4   +   1
```

packs both values into one 2-byte integer. If the value for the "high" register is 128 or greater, then after multiplying times 256 we must as usual subtract 65536 (to prevent "integer overflow")



MODEL I GOODIES

by Lance Wolstrup

Try this short BASIC program on your Model I (or in the Model I emulator) and you'll know just what Jerry Lee Lewis was singing about.

```
10 I$=INKEY$
20 OUT 255,8
30 OUT 255,0
40 IF I$="" THEN 10
```

It does produce a rather interesting display. And, NO, it doesn't work on the Model III, this program is for Model I only.

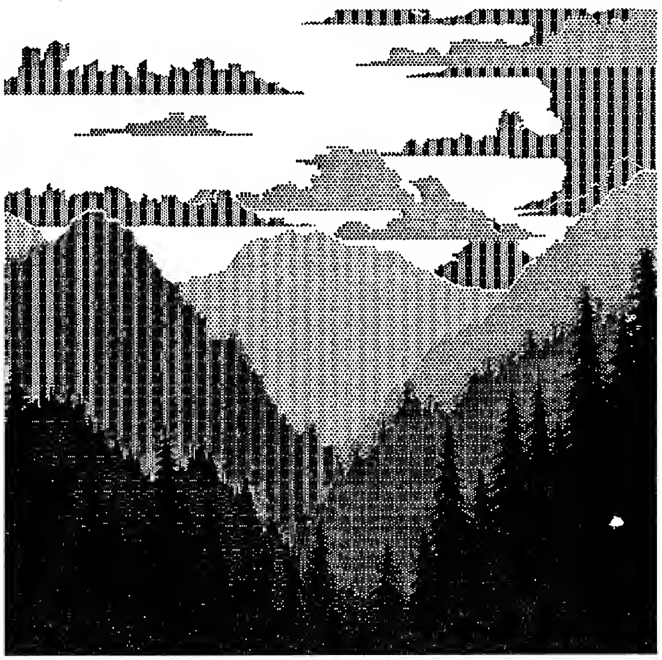
Another goodie is the hidden copyright message in TRSDOS. To see it, type the following directly from DOS Ready:

BOOT/SYS.WHO

and press ENTER. Now, while TRSDOS is looking for the file, hold down both the 2 and the **6** key. This will produce a full screen Radio Shack message you may never have seen before.

THE TOPANGA FIRE of 1993

by Jim King



Topanga Canyon, a "Y" shaped area named for a tribe of Indians who once lived there, is in the mountains 21 miles West of downtown Los Angeles, just inland and NE of coastal Malibu. One of the highest of the mountains in the canyon is 2800 feet (860 meters). The coast in that section runs nearly East and West, rather than North and South. Topanga Canyon Boulevard, which snakes through Topanga Canyon, stretches for about 20 miles from the sea to what's known as the Simi Freeway.

On Nov. 2, 1993 a series of devastating fires broke out in and around Topanga Canyon. It made headlines around the world. CNN showed dramatic shots of "Topanga Canyon" in flames. TRSTimes (and Lance Wolstrup) reside on Topanga Canyon Boulevard.. Fortunately, Lance is in Woodland Hills down on the "flatlands" of the San Fernando Valley, more than 4 miles (6.5 Km) north of the origin of the fire.

I wasn't quite as safely situated as TRSTimes since I happen to live deep in the heart of Topanga Canyon itself and very near the fire's origin. My house is end of a 150 foot driveway on a ridge with 4 other homes off what's known as Medley lane, off Tuna Canyon Road, in the town of Topanga. My house is stucco (i.e. covered with a layer of wire & cement), and all the others are wood. All homes in the canyon are required to have fire resisting roofs.

Because I live in what is known to be a "fire area," I put in some volunteer on Arson Watch. By the end of October, conditions were growing ripe for a fire. It was hot. It was dry. Winds, especially Santa Ana winds, could spell trouble. On October 26th the 'Santa Ana' wind (up to 50 mph) from the North did start up, the humidity dropped, and a 'Red Flag' was declared. Allen Emerson, the leader of the Arson Watch, called out the volunteers. I picked up a transceiver and AW signs at the fire station and drove to Mulholland Highway, named for the man who, ironically, pioneered the delivery of water to Los Angeles. About a half hour later smoke was seen in the direction of Thousand Oaks, west of us. The winds whipped up that fire and it burned for two straight days, all the way from Thousand Oaks to the ocean. We thought that was the end of it.

Tuesday, November 2nd, was election day, and I had planned to change the power input to a house down on Topanga Canyon Blvd. (I'm an electrician), then go to my friend Marco's house (the polling place) and vote. At 11 in the morning I got a call telling me there was a new fire, this one in Old Topanga Canyon (off Topanga Canyon). I could see a lot of smoke over the ridge to the NW. I postponed the job and started loading my van with valuables: all my tools, 2 of my 3-4P computers, a hard drive, a floppy drive, about 4 feet of computer manuals, electronic tools, heavy jackets, clothes, money. All the while I kept my eye on the TV to follow the progress of the fire. It had already burned up dozens of homes.

My next door neighbor had left for work. At this point, no one was being allowed back into the fire area. I phoned him and, at his request, I went over and got his files, keys, a video camera, and a bag of something, and put them in my van along with my things.

Bonnie, my neighbor to the north, an excitable woman anyway, was near hysterics trying to pack things and deal with her three children and a big unruly dog. I helped her get her papers and computer files, and she hastily left for relatives in Pasadena.

By Tuesday evening the fire had reached the sea, at which point it turned both East and West. The Eastern branch of the fire traveled along the beach to the beginning of Topanga Canyon Blvd., threatening to jump across. It also came up Tuna

Canyon, uncomfortably close to my house. It was burning brightly. At the rate it was moving East I thought it would reach my house before 11pm. I did not expect my house to survive if the fire came anywhere close since I have too many trees, too close by.

I called my ex-wife, Anna-Kria. She wanted me to make sure to takes our daughters' paintings if I had to evacuate. My plan was to wait until the sheriff came and told me to leave. I thought that when I saw the flames come over the ridge, it would be time to leave. I could see that the sky was red from the flames below.

Another neighbor put two hoses on his roof, turned the water on full blast -- and then he left. It was 10 pm. That angered me so much that I turned one of the hoses off and turned the other down to a trickle.

Now the three houses near me on the ridge were empty. I called my ex again at one in the morning. The fire beyond the West ridge looked like it would come over the crest at any moment, so I decided to evacuate to her house, even though the wind had died down somewhat.

At about 3 a.m. I put my cat 'Blackie' in the van and left. I drove slowly down Tuna Canyon to Topanga Canyon Boulevard. On the way I met firemen walking up the road and was told I was the last one to leave the area. Apparently there had been an evacuation order earlier and I hadn't heard it! There must have been 50 firetrucks parked EVERYWHERE!

I drove down TCB to my daughter and son-in-law's house. I knocked on their window: "Eve! Dan! Time to Go!" From her living room we could see the flames behind the ridge. They had already loaded their possessions in their camper/pickup and car.

I left and drove on down TCB, discovered it was blocked to the South, so I turned and drove slowly north again on the eerily deserted canyon road to the San Fernando Valley, then on to the freeway and East to Hollywood and Anna-Kria's house. Blackie howled the whole way. Eve and Dan arrived early the next morning.

On Wednesday the fires were still going. We watched TV most of the day. A friend told me they'd seen a shot of my house and it was still standing. When we phoned and got my answering machine we figured it was still there.

On Thursday morning November 4th Eve was permitted to go back to her house, but my ridge was still considered to risky. There was a police road block on TCB at Mulholland, another one just over the top of the ridge, and another at the intersection of Old and New TCB.

At 10 o'clock Thursday night I was finally given permission to return home. But when I rolled down my window to ask a fireman a question, Blackie jumped out. I was frantic. I spotted my friend Marco, gave him a flashlight and the two of us went running around the intersection calling 'KITTY, KITTY'. Finally Marco saw Blackie in his light, and I managed to grab her and stuff her back in the van. Relief! Once home, I got Blackie inside and fed, unloaded a few things and went to bed. I slept late.

Friday November 5th: I learned that the fire had come down the west ridge to nearby Tuna Canyon Road in three separate places, but had stopped there. No houses in that area were lost. That night I went out on Arson watch again.

Saturday November 6th: I drove down Stunt Road, East on Mulholland Highway, back South on Old Topanga Road, past the point of origin of the fire, ground zero. We learned that the fires had been started by an arsonist. Our Arson Patrols hadn't been enough. Now with so many houses vacated, the area was ripe for looting, but the CHP (California Highway Patrol) were everywhere, checking everyone coming in. As I drove on, I saw a firetruck from Barstow, 50 miles Northeast, then a firetruck from Redding, more than 500 miles North. There were even firefighters, including some women, from other states.

Sunday November 7th: The fires were mostly out although helicopters were still dropping sea water on a few hot spots.

Wednesday the 10th: We had a light rain, about 3/4 inch. It turned the grey burned hillsides to black. In Malibu, which only a week earlier had been in flames, people now had flooding and mud slide problems. It never stops!

** If you can get AAA Maps of Los Angeles & Vicinity I am at G4 and you can see where I am in relation to the city, or Ventura County (N11). If you cannot, send me SASE (Self Addressed Stamped Envelope) and I will return a copy of the Ventura map.*

Topanga Canyon Blvd. (TCB) goes up the Eastern branch of the canyon, Old Topanga Road goes up the West branch. Fernwood is the area Southwest of the intersection of Old & New Topanga roads.

SMART PRINTERS

by Roy T. Beck

I'm sure all of you have heard of smart terminals vs. dumb terminals, but I am using these adjectives to separate printers into two categories.

Actually, it is difficult to determine whether any given printer is a "smart printer" or a "dumb printer". There is no neat dividing line between, them, as the smart printers have gradually evolved over the years, and did not suddenly appear, full blown, on the market. In fact, I don't think I will even try to classify them, this will be left to the student as an exercise, as one of my old profs liked to say.

I will hark back to my oldest printer, an NCR thermal dot matrix printer. It was, by any standard, a DUMB printer. Its greatest virtue was that it was very quiet in an era when impact type dot matrix printers would fairly scream at you, driving persons with tender ears out of the room. This NCR used thermal paper, and formed letters by selectively heating dots on the face of the printhead. The heating process was silent; its only noise was caused by "patting" the head against the paper for a few milliseconds. But that printer was S L O W, (10 characters per second) its thermal paper with blue characters was prone to fading with age, the paper was only 8.2 inches wide, and it had other tiresome features. It had never heard of a form feed. But it was my first printer, and I still have fond remembrances of it. After all, if your computer images cannot be recorded on paper, what good are the results?

My next printer was the ubiquitous EPSON MX-80/FT. This one began to take toddling steps up the ladder to intelligence. Bear in mind, EPSON did put some nice features in this printer, including the ability to underline, italicize, print condensed and double wide, double strike, emphasize, and print 8 lines per inch. But the only way to access these features was to send certain control code sequences from the computer. If you were in the middle of some other program, there was no convenient way to insert the necessary control code from the keyboard.

But not all of its ultimate smarts were courtesy of the factory. How many of you remember Dan Dresselhaus, a former (and I believe, founding member) of SAGATUG, the San Gabriel Valley TRS users group, one of the oldest TRS groups around? Well, Dan also had an MX-80, and he very soon became excessively aggravated by the fact that BASIC listings just ran on and on, spilling over from one page to another, without regard to the precut tear lines on the fan-fold paper. The usual result was loss of one line of BASIC when you LLISTed your program, and a mess when you tried to save the printout in a binder. But, besides cussing, Dan did something about it. He first found the documentation for the machine code embedded in the ROM in the MX-80. Next, he wrote a disassembler for that language, and proceeded to disassemble the MX-80 ROM contents and document all the code. He then worked out a way to modify the factory code in such a way that the machine could be, optionally, told to leave 6 blank lines after every 60 print lines.

The technique which Dan invented was to allow the built-in features of the printer to be accessed by manipulating the four buttons mounted on top of the printer. By taking the machine off-line, and doing a little finger twiddling on the buttons, you could turn on or off, selectively, the features the factory had built in, plus you could control some additional features added by DAN.

Presto, Dan had invented an after-market package named PERF-SKIP, which, in addition to the factory supplied features, would skip over the perforations and allow a three blank line header at top and bottom of each page. He began selling this device at local club meetings, and it was an instant success. It has since gone through several name changes, the last one being, I think, "FingerPrint". Shortly after its introduction, yours truly advised him to add another optional feature, that being a 1/2" left hand margin to allow three-hole punching of the pages without punching out the line numbers. (Since this small feature is about my only claim to fame, I had to work it in here!) Still other features rapidly sprang from Dan's fertile imagination, and soon he was literally in business for himself, which he continues to be today, under the name (I believe) of Dresselhaus Industries, Inc.

But to continue with printer intelligence, let me describe some of the other features added by Dan and others.

At some later date, the concept of Near Letter Quality dot matrix printing came about. I don't know who actually invented it, but again Dan did some inspired thinking and imagining. The MX-80 was by then superseded by numerous other models, but by golly, there were an awful lot of them still in use. One of its shortcomings was lack of a NLQ

mode. Dan next dreamed up a replacement ROM scheme which would allow the user to switch from the factory default type face to an NLQ face which was a vast improvement!

Another feature added by manufacturers was the ability to print in both directions as the head flew back and forth from left to right, right to left, etc. Implementing this bi-directional printing required a small internal buffer in the printer. From here it was only a small step to adding large aftermarket buffers to printers, either inside or external to the printer. The earliest aftermarket buffers I know about were only 16K, which would hold typically 3 or 4 pages of text. Later ones got much larger, up to 1 Meg, I believe.

Back to Dan for a moment. With some printer buffers, there was no convenient way to empty the contents if you suddenly realized there was a typo in the middle of a 10 page document, and you no longer wanted to print the document. Dan to the rescue! He added another feature to his keypad control system, which caused the printer to accept the contents of the buffer at a very high rate of speed, but didn't print anything. Thus, the buffer was emptied in a matter of a few seconds, and the user could proceed to his corrections, etc.

Unfortunately for Dan, all the features he added to the EPSON family (and to certain other printers) were not eligible for patent protection. Fortunately for all the rest of us users, the manufacturers saw the utility of Dan's fundamental idea (making control accessible on the printer keypad) and promptly incorporated these features into their future models. The result was a significant improvement in printer intelligence, to the advantage of all of us. Don't feel too badly about Dan, as his imagination continued to come up with other ideas, and he continues in business for himself.

After the dot matrix and daisy wheel printers came the two latest entries in the evolving printer world. These are the ink jet and the laser printers. The inkjet printer uses electrostatic attraction to squirt ink though very fine apertures in the printhead onto the paper. This can be done with great accuracy and controllability, yielding nicely formed characters without physical impact between the printhead and the paper. Just don't ask for carbon copies! The latest ink jets can even print in multiple colors,

And of course, we now have laser printers. Laser printers actually contain a specialized computer with a significantly large memory capacity, and most of the essential components of a Xerox copying machine. Where a copier scans a document and uses the reflected light from the document to selectively discharge the surface of a statically charged drum, the laser printer uses a laser light source controlled by the internal computer to selectively discharge the charged drum. In either case, the partially discharged drum then picks up toner particles from a supply and transfers them to the output paper, After which they are fused in place to form a very permanent image on the paper. If your pockets are deep enough, you can have laser printing in colors, and quite good ones, at that. In fact, the latest Xerox copiers are so good, they are actually being used to print counterfeit paper money. This has required the US Treasury to make some significant changes in the design of our paper money. But that's another story for another day.

In conclusion, there has been a remarkable improvement in the I.Q. of our printers over the last ten or so years, but just you wait! Take a careful look back by the year 2000 and you will see things not even dreamed of today.

Anticipating the future, I remain your faithful scribe.

Roy.

HINTS & TIPS

TRS-80 SPEAKERS

by Kelly Bates

I have speakers in all my computers and thought you might like to know how and why. Some folks zap their games so the correct sound will be sent to the right port, but I prefer the speaker installed. That way my disk programs remain original and always work as written.

The Model III has no sound, so first get a speaker and mini-switch, some wire and 2 mini alligator clips. Wire a clip to one side of the speaker (8 ohm), wire the switch to the other side of the speaker with the other clip at the end of the wire. Now you have a test circuit.

Open the Model III and expose the CPU board. Find U78. R21 (7.5 meg) and R20 (220K) both connect to pin 6 of U78. Connect your circuit to that junction, and connect the other end of your circuit to ground. With the electrical connections made on the computer, boot it up and load a program that has sound. If you do not hear sound, toggle your switch. When you have it working, shut everything down and mount your switch next to the Cass plug on the rear of the computer. Mount the speaker anywhere. I glued mine to the bottom of the computer. Yes, you will have to drill a hole for the switch to mount into. You are tapping into the Cass-Out circuit, and the switch lets you normal up the computer so you can use the cassette recorder. Also lets you turn off the sound. So much for sound in the Model III.

In the 4P, remove the transducer and disconnect one end of R19 (22 ohms). Put in a speaker and volume adjust in lieu of R19. The 4P has no cassette circuit, so this mod is straight forward. I mounted my speaker in the cavity behind the drives.

Standard Model 4 - Expose the CPU and locate U27 pin 8, R55 (7.5K) and R29 (220K). Connect your test circuit to that junction and see how your sound works. That is again the Cass-Out circuit, like the Model III, so you will need the switch to re-enable the Cass-Out circuit for its original function. You now have two sound circuits, and at least one will work in most operations without having to zap the software.

The 4D should be the same, but I have no drawings for it. The 4D also has a separate speaker board, so you will have two sound sources there when you finish.

MODEL I EMULATOR & DOUBLE DENSITY DISKS

by Lance Wolstrup

There has been much discussion as to "why the Model I emulator cannot read double-density disks." First of all, the question is bogus. The emulator does not read disks — it reads files — and files have no density. Are you with me so far?

Now, Mr. Vavasour hard-wired the emulator to expect the file (virtual disk) to begin the directory at record 170 (AAh). 'Why at record 170?', you might ask. Because record 170 is the product of the number of tracks and sectors it takes to get to the directory on a standard Model I single-density disk (17 tracks times 10 sectors). Incidentally, that is why some Model I LDOS or DOSPLUS disks will not work on the emulator — they aren't standard — the directory is probably on track 20!

We have now established that single-density disks with the directory beginning at track 17, sector 0 will work correctly. So, the question now becomes 'What if we transfer a double-density disk to an emulator file?' The immediate answer is that it won't work because the directory will be in the wrong place; that is, it will be located at record 306 (17 tracks times 18 sectors). But if we leave our thinking caps on a bit, we would come to the realization that if we could somehow manage to place the directory so it would begin at record 170, rather than 306, it just might work!

Trying to convert the position of the directory to another place is not a programming exercise that I recommend, so we are fortunate that we don't have mess with it. You see, NEWDOS/80 v2 has solved the problem for us. Believe it or not, a standard Model I/III NEWDOS/80 v2 double-density disk begins its directory on track 9, sector 8. If you do a little calculation, you will see that 9x18+8 does indeed equal 170 — the directory begins at record 170 — just what the emulator expects.

I have transferred most of my NEWDOS/80 Model III (double-density) disks to the emulator, and they work perfectly (do note that the programs will only work if they are Model I compatible). Transferring the double-density disks to virtual files was an easy task. I simply patched my MAKFILE/CMD and MAKFILE4/CMD programs which were listed in our last issue to work with double-density disks. I would recommend that you copy MAKFILE/CMD to a new file called DD2FILE/CMD, and copy MAKFILE4/CMD to DS42FILE/CMD. Then patch the two new files as follows:

```
PATCH DD42FILE/CMD (D01,93=12:F01,93=0A)
PATCH DD42FILE/CMD (D01,B3=12:F01,B3=0A)
```

This alters the Model 4 version of the program to read and write 18 sectors per track, instead of 10.

Now, if you prefer to use your Model I or III to transfer the double-density disks to virtual files, you will need to patch DD2FILE/CMD in the same manner. If you use LDOS 5.3.x to patch, type:

```
PATCH DD2FILE/CMD (D01,A2=12:F01,A2=0A)
PATCH DD2FILE/CMD (D01,C2=12:F01,C2=0A)
```

If you are a Model I/III NEWDOS/80 fan, you will have to use SUPERZAP to modify the file. Enter SUPERZAP and choose:

DFS

Answer the FILESPEC? prompt with:

DD2FILE/CMD.

Answer RELATIVE-SECTOR-WITHIN-FILE # prompt by typing: **1**

Now type **MOD A2**

The cursor appears on top of the 0A at byte A2. Type **12**.

Use the arrow keys to move the cursor down to byte C2. There, overstrike the 0A by typing **12**.

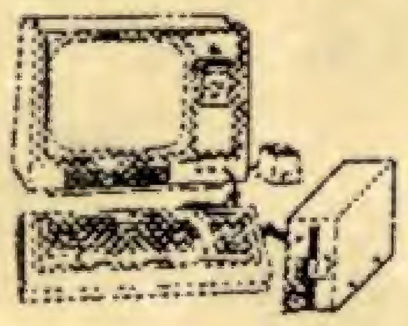
After this is done, press **ENTER**.

SUPERZAP will ask you if it is OK to write the changes to disk. Answer by pressing **Y**.

Pressing ENTER returns you to the newly modified record. Type **EXIT** and you are back in DOS.

If you have double-density disks that you want to transfer to emulator virtual disk files, you can now use NEWDOS/80 as the transfer vehicle. Simply copy all files from whatever DOS they were on over to NEWDOS, and then use DD42FILE/CMD to read the disk and write the file (from Model 4), or use DD2FILE/CMD to do the same from Model I/III LDOS. Sorry, but as explained in last months article, MAKFILE/CMD and DD2FILE/CMD will only work from LDOS, as it is the only DOS that, to my knowledge, has documented its RDSEC, RDSSEC, WRSEC and WRSSEC routines. I simply have not had time to go hunting for the addresses in NEWDOS — and they ARE different than in LDOS. You will just have to live with that — unless you know what the NEWDOS addresses are. In that case, please write a short article and let me (and subsequently, the readers) know.

Meanwhile, enjoy your double-density disks running on the Model I emulator. And they said it couldn't be done. Ha!!



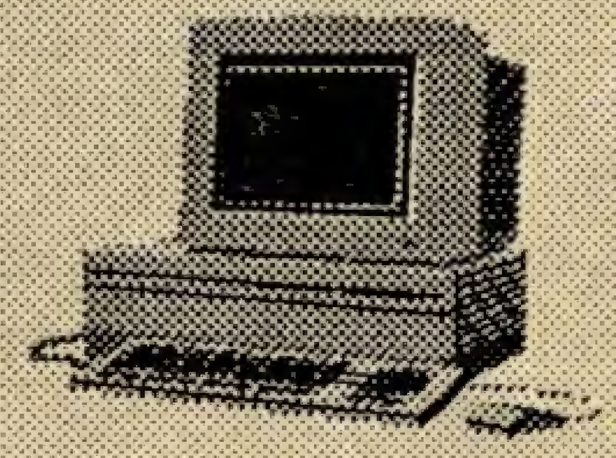
ITEMS OF INTEREST

VTUG

Valley TRS-80 Users Group
meets the third Friday of each month
at 7:30 pm at the Learning Tree University.
DeSoto & Knapp Streets, Chatsworth, CA.
Contact Lance Wolstrup (818) 716-7154

Mid-Cities Tandy Radio Shack Users Group

MCTRUG supports all of the Tandy computers plus IBM compatibles. We have software available for TRS-80 Models I, III, 4, Color computers, Model 2000 and MS-DOS.



Write us for more information. Please include your name, mailing address, computer model and which Disk Operating System you use.

Write to: MCTRUG
P.O. Box 171566. Arlington, TX 76003

VTHG

Valley TRS-80 Hackers Users Group
Los Angeles, CA.
meets the first Friday of each month
at 7:00 pm.
For more information
contact Roy Beck (213) 664-5059

SAGATUG

San Gabriel Tandy Users Group
meets the second Friday of each month at
7:00 pm at the Arcadia Park Senior
Citizen's Center. 405 S. Santa Anita Ave.
Arcadia, CA
THE club for TRSDOS, MS-DOS, CP/M
and Laptop computers.
Large public domain libraries
Remote memberships available
Contact Barbara Beck (213) 664-5059